Face Recognition Challenges and Evaluations (FaCE)

APIs for Recognition and Detection of Faces in 2D Still Images

Version 1.0

Patrick Grother and Mei Ngan

Image Group
Information Access Division
Information Technology Laboratory

National Institute of Standards and Technology
U.S. Department of Commerce

July 30, 2015

Table of Contents

1. FaCE 3
1.1. Options for participation 3
1.2. Schedule 3
1.3. Hardware specification 3
1.4. Operating system, compilation, and linking environment 3
1.5. Software and Documentation 4
1.6. Runtime behavior 5
1.7. Threaded computations 5
2. Data structures supporting the API 6
2.1. Overview 6
2.2. Requirement 6
2.3. Data structures 6
3. API 7
3.1. 1:1 Verification 7
3.2. Face Detection 10
Annex A Submission of Implementations to the FaCE 11
A.1 Submission of implementations to NIST 11
A.2 How to participate 11

List of Figures

Figure 1 – Schematic of verification 8

List of Tables

Table 1 – FaCE classes of participation 3
Table 2 – Implementation library filename convention 4
Table 3 – IMAGE struct 6
Table 4 – MULTIFACE typedef 6
Table 5 – BOUNDING_BOX struct 6
Table 6 – Functional summary of the 1:1 application 7
Table 7 – Implementation template size requirements 8
Table 8 – SDK initialization 8
Table 9 – Template generation 9
Table 10 – Template matching 9
Table 11 – SDK initialization 10
Table 12 – Face detection 10

1. FaCE

1.1. Options for participation

The following rules apply:

- A participant must properly follow, complete and submit the Annex A Participation Agreement. It is not necessary to do this for each submitted software submission to NIST, hereafter referred to as "SDK".
- Any SDK shall implement exactly one of the functionalities defined in Table 1. So, for example, the face detection functionality of a class DETECTION SDK shall not be merged with that of a class 1to1 SDK.
- In cases of corporate merger, or acquisition, or significant change in management, NIST may require re-submission of a properly completed application form.

Table 1 – FaCE classes of participation

Function	1:1 verification	Face Detection
Class label	1to1	DETECTION
API requirements	3.1.2 + 3.1.3 + 3.1.4 + 3.1.5	3.2.1 + 3.2.2

1.2. Schedule

Participants may submit up to two algorithms at their first entry to the FaCE 1:1 evaluation. Thereafter, organizations may submit algorithms no sooner than 150 calendar days after the prior submission.

In maintaining the public results reports, NIST will generally maintain results for the two most recently submitted algorithms. NIST will generally cease to maintain results for algorithms submitted more than three years ago. NIST may maintain additional results, particularly if they have some notable aspect (e.g. small template size).

1.3. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The blades are labeled as Dell M905, M910, and M610. The following list gives some details about the hardware of each blade type:

- Dell M610 - Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each)
- Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs[1] (4 cores each)
- Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)

Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. We anticipate that 16 processes can be run without time slicing.

NIST is requiring use of 64 bit implementations throughout.

1.4. Operating system, compilation, and linking environment

The operating system that the submitted implementations shall run on will be released as a downloadable file accessible from http://nigos.nist.gov:8080/evaluations/, which is the 64-bit version of CentOS 7.0 running Linux kernel 3.10.0.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux.

NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library in a format that is linkable using the C++11 compiler, g++ version 4.8.2.

A typical link line might be

[1] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3wext 3dnow constant_tsc nonstop_tsc pni cx16 popcnt lahf_lm cmp_legacy svm extapic cr8_legacy altmovcr8 abm sse4a misalignsse 3dnowprefetch osvw

g++ -I. -Wall -m64 -std=c++11 -o facechallengestest facechallengestest.cpp -L. –lfacechallenges_fmitga_1to1_07

The Standard C++ library should be used for development of the SDKs. The prototypes from the API of this document will be written to a file "facechallenges.h" which will be included via

```
#include <facechallenges.h>
```

The header files will be made available to implementers at http://nigos.nist.gov:8080/facechallenges/src.

NIST will handle all input of images via the JPEG and PNG libraries, sourced, respectively from http://www.ijg.org/ and http://libpng.org.

All compilation and testing will be performed on x86 platforms. Thus, participants are strongly advised to verify library-level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.). It is highly recommended that participants run the open MEDS 1:1 challenge, available for download from http://www.nist.gov/itl/iad/ig/facechallenges.cfm.

Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

1.5. Software and Documentation

1.5.1. SDK Library and Platform Requirements

Participants shall provide NIST with binary code only (i.e. no source code). Header files (".h") are allowed, but these shall not contain intellectual property of the company nor any material that is otherwise proprietary. The SDK should be submitted in the form of a dynamically linked library file.

The core library shall be named according to Table 2. Additional shared object library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

Intel Integrated Performance Primitives (IPP) libraries are permitted if they are delivered as a part of the developer-supplied library package. It is the provider's responsibility to establish proper licensing of all libraries. The use of IPP libraries shall not inhibit the SDK's ability to run on CPUs that do not support IPP. Please take note that some IPP functions are multithreaded and threaded implementations may complicate comparative timing.

SDKs will not have access to graphics processing units (GPUs).

Table 2 – Implementation library filename convention

Form	libfacechallenges_provider_class_sequence.ending				
Underscore delimited parts of the filename	libfacechallenges	provider	class	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: fmitga	"1to1" for 1:1 submissions "DETECTION" for face detection submissions	A two digit decimal identifier to start at 00 and increment by 1 every time any SDK is sent to NIST. EXAMPLE: 07	.so
Example	libfacechallenges_fmigta_1to1_07.so				

1.5.2. Configuration and developer-defined data

The implementation under test may be supplied with configuration files and supporting data files. The total size of the SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes = 1024^3 bytes.

1.5.3. Submission folder hierarchy

Participant submissions should contain the following folders at the top level

- lib/ - contains all participant-supplied software libraries
- config/ - contains all configuration and developer-defined data
- doc/ - contains any participant-provided documentation regarding the submission

1.5.4. Installation and Usage

The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The SDK shall be installable using simple file copy methods. It shall not require the use of a separate installation program.

The SDK shall neither implement nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The submitted implementations shall remain operable with no expiration date.

Hardware (e.g. USB) activation dongles are not permitted.

1.6. Runtime behavior

1.6.1. Interactive behavior

The SDK will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall not use graphical user interfaces nor terminal interaction e.g. from “standard input”.

1.6.2. Error codes and status messages

The SDK will be tested in non-interactive “batch” mode, without terminal support. Thus, the submitted library shall run quietly, i.e. it should not write messages to "standard error" and shall not write to “standard output”. An SDK may write debugging messages to a log file - the name of the file must be declared in documentation.

1.6.3. Exception Handling

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

1.6.4. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process), nor read from such. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.6.5. Stateless behavior

All software components shall be stateless. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.7. Threaded computations

Threading is not permitted, because NIST will parallelize the test by dividing the workload across many cores and many machines. NIST's calling applications are single-threaded.

2. Data structures supporting the API

2.1. Overview

This section describes separate APIs for the classes of participation described in section Table 1. All SDK's submitted to FaCE shall implement the functions required by the rules for participation listed in section 1.1.

2.2. Requirement

FaCE participants shall submit an SDK which implements the relevant C++ prototyped interfaces of clause 3. C++ was chosen in order to make use of some object-oriented features.

2.3. Data structures

2.3.1. Data structure for encapsulating a single still image

For face verification, an individual is represented by K ≥ 1 two-dimensional facial images. All facial images in the verification test will contain one and only one face per image. For the face detection task, an image may contain one or more faces.

2.3.2. Struct for encapsulating a single image

Table 3 – IMAGE struct

	C++ code fragment	Remarks
1.	`typedef struct IMAGE`	
2.	`{`	
3.	`uint16_t width;`	Number of pixels horizontally
4.	`uint16_t height;`	Number of pixels vertically
5.	`uint8_t depth;`	Number of bits per pixel. Legal values are 8 and 24.
6.	`uint8_t format;`	Flag indicating native format of the image as supplied to NIST 0x01 = JPEG (i.e. compressed data) 0x02 = PNG (i.e. never compressed data)
7.	`uint8_t *data;` `} IMAGE;`	Pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes `IIIIIIII`

2.3.3. Typedef for encapsulating a set of face images from a single person

Table 4 – MULTIFACE typedef

	C++ code fragment	Remarks
1.	`typedef std::vector<IMAGE> MULTIFACE;`	Vector containing F pre-allocated face images of the same person. The number of items stored in the vector is accessible via the vector::size() function.

2.3.4. Struct for encapsulating a detected face from an image

Table 5 – BOUNDING_BOX struct

	C++ code fragment	Remarks

1.	`typedef struct BOUNDING_BOX`
	`{`
	`uint16_t x;`
		x-coordinate of top-left corner of bounding box around face

2.	uint16_t y;	y-coordinate of top-left corner of bounding box around face
3.	uint16_t width;	width, in pixels, of bounding box around face
4.	uint16_t height;	height, in pixels, of bounding box around face
5.	} BOUNDING_BOX;	

2.3.5. Data type for similarity scores

Verification functions shall return a measure of the similarity between the face data contained in the two templates. The data type shall be an 64-bit double precision real. The legal range is [0, DBL_MAX], where the DBL_MAX constant is larger than practically needed and defined in the <limits.h> include file. Larger values indicate more likelihood that the two samples are from the same person.

Providers are cautioned that algorithms that natively produce few unique values (e.g. integers on [0,127]) will be disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly 0.0001, for example.

3. API

3.1. 1:1 Verification

3.1.1. Overview

The 1:1 testing will proceed in three phases: preparation of enrollment templates; preparation of verification templates; and matching. These are detailed in Table 6.

Table 6 – Functional summary of the 1:1 application

Phase	Description	Performance Metrics to be reported by NIST
Initialization	Function to allow implementation to read configuration data, if any.	None
Enrollment	Given K ≥ 1 input images of an individual, the implementation will create a proprietary enrollment template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Verification	Given K ≥ 1 input images of an individual, the implementation will create a proprietary verification template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Matching (i.e. comparison)	Given one proprietary enrollment template and one proprietary verification template, compare these and produce a similarity score. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs.

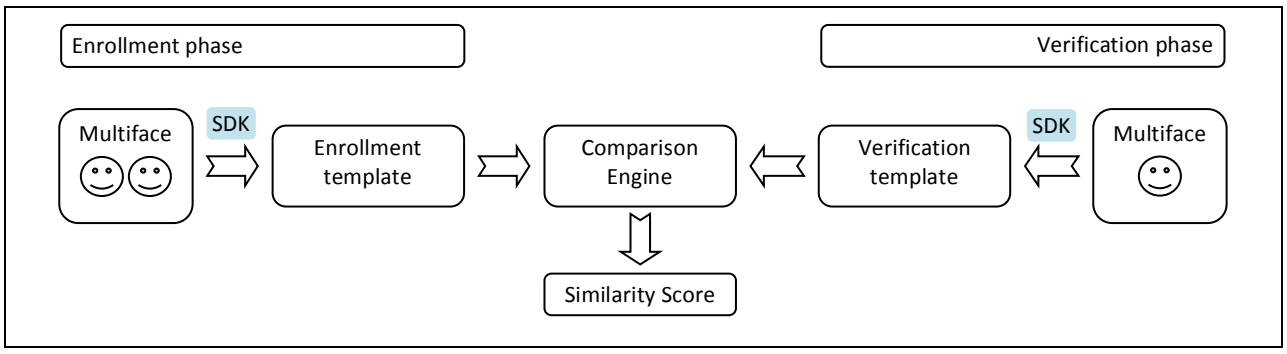

Figure 1 – Schematic of verification

3.1.2. Maximum template size

All implementations shall report the maximum expected template sizes. These values will be used by the NIST test harnesses to pre-allocate template data. The values should apply to a single image. For a **MULTIFACE** containing K images, NIST will allocate K times the value returned. The function call is given in Table 7.

Table 7 – Implementation template size requirements

Prototype	int32_t get_max_template_sizes(	
	uint32_t &max_enrollment_template_size,	Output
	uint32_t &max_verification_template_size);	Output
Description	This function retrieves the maximum template size needed by the feature extraction routines.	
Output Parameters	max_enrollment_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single enrollment image.
	max_verification_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single verification or identification image.
Return Value	0	Success
	Other	Vendor-defined failure

3.1.3. Initialization

Before any template generation or matching calls are made, the NIST test harness will make a call to the initialization of the function in Table 8.

Table 8 – SDK initialization

Prototype	int32_t initialize_verification(	
	const std::string &configuration_location);	Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the Table 9 functions convert_multiface_to_enrollment_template or convert_multiface_to_verification_template. The SDK under test should set all parameters.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
Output Parameters	none	
Return Value	0	Success
	2	Vendor provided configuration files are not readable in the indicated location.
	Other	Vendor-defined failure

3.1.4. Template generation

The functions of Table 9 support role-specific generation of a template data. The format of the templates is entirely proprietary.

Table 9 – Template generation

Prototypes	int32_t convert_multiface_to_enrollment_template(		
	const MULTIFACE &input_faces,		Input
	uint32_t &template_size,		Output
	uint8_t *proprietary_template);		Output
	int32_t convert_multiface_to_verification_template(		
	const MULTIFACE &input_faces,		Input
	uint32_t &template_size,		Output
	uint8_t *proprietary_template);		Output
Description	This function takes a MULTIFACE, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. In all cases, even when unable to extract features, the output shall be a template record that may be passed to the match_templates function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this.		
Input Parameters	input_faces	An instance of a Table 4 structure. Implementations must alter their behavior according to the number of images contained in the structure.	
Output Parameters	template_size	The size, in bytes, of the output template	
	proprietary_template	The output template. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the MULTIFACE; the value T is output by the maximum template size functions of Table 7.	
Return Value	0	Success	
	2	Elective refusal to process this kind of MULTIFACE	
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)	
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)	
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)	
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.	

3.1.5. Matching

Matching of one enrollment against one verification template shall be implemented by the function of Table 10.

Table 10 – Template matching

Prototype	int32_t match_templates(		
	const uint8_t *verification_template,		Input
	const uint32_t verification_template_size,		Input
	const uint8_t *enrollment_template,		Input
	const uint32_t enrollment_template_size,		Input
	double &similarity);		Output
Description	This function compares two opaque proprietary templates and outputs a similarity score, which need not satisfy the metric properties. NIST will allocate memory for this parameter before the call. When either or both of the input templates are the result of a failed template generation (see Table 9), the similarity score shall be -1 and the function return value shall be 2.		
Input Parameters	verification_template	A template from convert_multiface_to_verification_template().	
	verification_template_size	The size, in bytes, of the input verification template $0 \le N \le 2^{32} - 1$	
	enrollment_template	A template from convert_multiface_to_enrollment_template().	
	enrollment_template_size	The size, in bytes, of the input enrollment template $0 \le N \le 2^{32} - 1$	
Output	similarity	A similarity score resulting from comparison of the templates, on the range	

Parameters		[0,DBL_MAX].
Return Value	0	Success
	2	Either or both of the input templates were result of failed feature extraction
	Other	Vendor-defined failure

3.2. Face Detection

3.2.1. Initialization

Before any calls to detect_faces are made, the NIST test harness will make a call to the initialization of the function in Table 11.

Table 11 – SDK initialization

Prototype	int32_t initialize_detection(	
	const std::string &configuration_location);	Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the function detect_faces. The SDK under test should set all parameters.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
Output Parameters	none	
Return Value	0	Success
	2	Vendor provided configuration files are not readable in the indicated location.
	Other	Vendor-defined failure

3.2.2. Face detection

The function of Table 12 supports the detection of faces in an image. An image may contain one or more faces.

Table 12 – Face detection

Prototypes	int32_t detect_faces(		
	const IMAGE &input_image,		Input
	std::vector<BOUNDING_BOX> &bounding_boxes);		Output
Description	This function takes an IMAGE as input, and populates a vector of BOUNDING_BOX with the number of faces detected from the input image. The implementation could call vector::push_back to insert into the vector.		
Input Parameters	input_image	An instance of a struct representing a single image from Table 3.	
Output Parameters	bounding_boxes	For each face detected in the image, the function shall create a BOUNDING_BOX (see Table 5), populate the x, y, width, height of the bounding box, and add it to the vector.	
Return Value	0	Success	
	2	Elective refusal to process this kind of IMAGE	
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)	
	6	Cannot parse input data	
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.	

Annex A
Submission of Implementations to the FaCE

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: http://www.nist.gov/itl/iad/ig/encrypt.cfm

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

A.2 How to participate

Those wishing to participate in FaCE testing must do all of the following.

- IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here. http://www.nist.gov/itl/iad/ig/encrypt.cfm
- Send a signed and fully completed copy of the *Application to Participate in the Face Recognition Challenges and Evaluations (FaCE)*. This is available at http://www.nist.gov/itl/iad/ig/facechallenges.cfm. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed FaCE Application to Participate shall be sent to NIST as a PDF.
- Provide an SDK (Software Development Kit) library, which complies with the API (Application Programmer Interface) specified in this document.
    - Encrypted data and SDKs below 20MB can be emailed to NIST at FaceSubmissions@nist.gov.
    - Encrypted data and SDKS above 20MB shall be

      EITHER
        - Split into sections AFTER the encryption step. Use the unix "split" commands to make 9MB chunks, and then rename to include the filename extension need for passage through the NIST firewall.
        - `you%  split -a 3 -d -b 9000000  libfacechallenges_fmitga_1to1_02.tgz.gpg`
        - `you%  ls -1 x??? | xargs -iQ  mv Q libfacechallenges_fmitga_1to1_02_Q.tgz.gpg`
        - Email each part in a separate email. Upon receipt NIST will
        - `nist%  cat facechallenges_fmitga_1to1_02_*.tgz.gpg > libfacechallenges_fmitga_1to1_02.tgz.gpg`

      OR
        - Made available as a file.zip.gpg or file.zip.asc download from a generic http webserver[2],

      OR
        - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

FaCE Test Liaison (A203) 100 Bureau Drive A203/Tech225/Stop 8940 NIST	In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296.

[2] NIST will not register, or establish any kind of membership, on the provided website.

Gaithersburg, MD 20899-8940 USA	

261